

ECE 741/841

17 October 2002

Ripple Carry Adder

- We have a Specification and High Level implementation of a ripple carry adder.
- The specification defines the behaviour of the adder represented by natural numbers.
- The high level implementation defines a one bit full adder.
- Using the one bit adder, and a N bit ripple carry adder is implemented.
- A notion of bits and bit vectors is defined in the theory digits
- All theories are parameterized with the length N of the bit vector (word length).

Verification

- The verification consists in showing that the Specification is equivalent to the High Level implementation.
- Since the Specification is defined in terms of natural numbers and the High Level implementation in terms of bits, a function is defined to convert bit vectors to their natural number equivalent.

Properties Theory

The properties theory contains the lemmas and theorem for the verification.

```
properties : theory
begin
```

```
N : above[0]
```

```
importing spec[N] , high_level_Impl[N]
```

```
a,b,cin,adder_cin : var bit
X,Y : var bvec
```

```
OBA_correct : LEMMA
2*OBA_out(a,b,cin) + OBA_sum(a,b,cin) = a + b + cin
```

```
adder_correct_rec : LEMMA
FORALL (n:below[N]):
```

```

n_cout(n,X,Y,adder_cin)*exp2(n+1) + bv2nat_rec(n,adder_sum(X,Y,adder_cin)) =  

bv2nat_rec(n,X) + bv2nat_rec(n,Y) + adder_cin

all_adder_correct : LEMMA
n_cout(N-1,X,Y,adder_cin)*exp2(N) + bv2nat(adder_sum(X,Y,adder_cin)) =  

bv2nat(X) + bv2nat(Y) + adder_cin

adder_correct_2 : LEMMA
n_cout(N-1,X,Y,adder_cin)*exp2(N) + bv2nat(adder_sum(X,Y,adder_cin)) =  

adder_cout(bv2nat(X),bv2nat(Y),adder_cin)*exp2(N) +  

adder_S(bv2nat(X),bv2nat(Y),adder_cin)

cout_correct_rec : LEMMA
FORALL (n:below[N]):  

n_cout(n,X,Y,adder_cin) =  

adder_cout_aux(bv2nat_rec(n,X),bv2nat_rec(n,Y),adder_cin,n)

cout_correct : THEOREM
n_cout(N-1,X,Y,adder_cin) = adder_cout(bv2nat(X),bv2nat(Y),adder_cin)

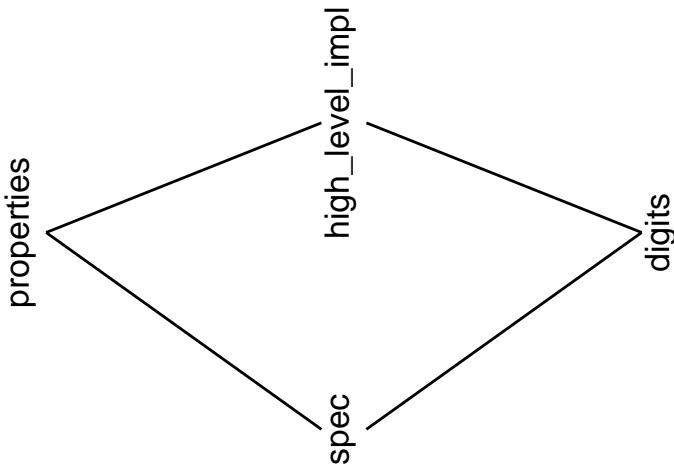
adder_correct : THEOREM

```

```
 bv2nat (adder_sum (X,Y,adder_cin)) =  
 adder_S (bv2nat (X),bv2nat (Y),adder_cin)  
 end properties
```

Theory Hierarchy

This graph represent the theory hierarchy. It can be generated by the PVS command M-x x-theory-hierarchy. It shows the theory dependencies; spec and `high_level_Impl` import digits. properties imports spec and `high_level_Impl`.

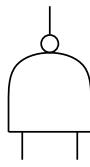


Gate Level Implementation

- The next step in our development is the definition of the gate level implementation.
- We have to define the functionality of gates.
- Since we have defined bits as zeros and ones {0,1}, we have two options to define gates:
`beginitemize`
- convert bits to true or false and use logical connectives,
or
- define the gates as functions.
`enditemize`

Gate Definitions

Example, NAND gate.



```
gates[N:above[0]] : theory
begin
importing digits[N]
```

```
a,b,c,d : var bit
p : var bool
```

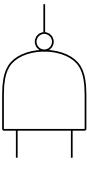
```
b2tf(a) : bool = (a = 1)
```

```
tf2b(p) : bit = if p then 1 else 0 endif
```

```
NAND2i(a,b) : bit = tf2b(NOT(b2tf(a) AND b2tf(b)))
```

Gate Definitions

Example, NAND gate.



```
a,b,c,d : var bit
```

```
NAND2i(a,b) : bit = if a=1 AND b=1 then 0 else 1 endif
```

Other Gate Definitions

```
a,b,c,d : var bit
p : var bool

dNOT(a) : bit = tf2b(NOT(b2tf(a)))

% dNOT(a) : bit = if a=0 then 1 else 0 endif

AND2i(a,b) : bit = tf2b(b2tf(a) AND b2tf(b))

% AND2i(a,b) : bit = if a=1 AND b=1 then 1 else 0 endif

OR2i(a,b) : bit = tf2b(b2tf(a) OR b2tf(b))

% OR2i(a,b) : bit = if a=1 OR b=1 then 1 else 0 endif
```

```

NAND2i(a,b) : bit = tf2b(NOT(b2tf(a) AND b2tf(b)))

% NAND2i(a,b) : bit = if a=1 AND b=1 then 0 else 1 endif

NOR2i(a,b) : bit = tf2b(NOT(b2tf(a) OR b2tf(b)))

% NOR2i(a,b) : bit = if a=1 OR b=1 then 0 else 1 endif

AND3i(a,b,c) : bit = tf2b(b2tf(a) AND b2tf(b) AND b2tf(c))

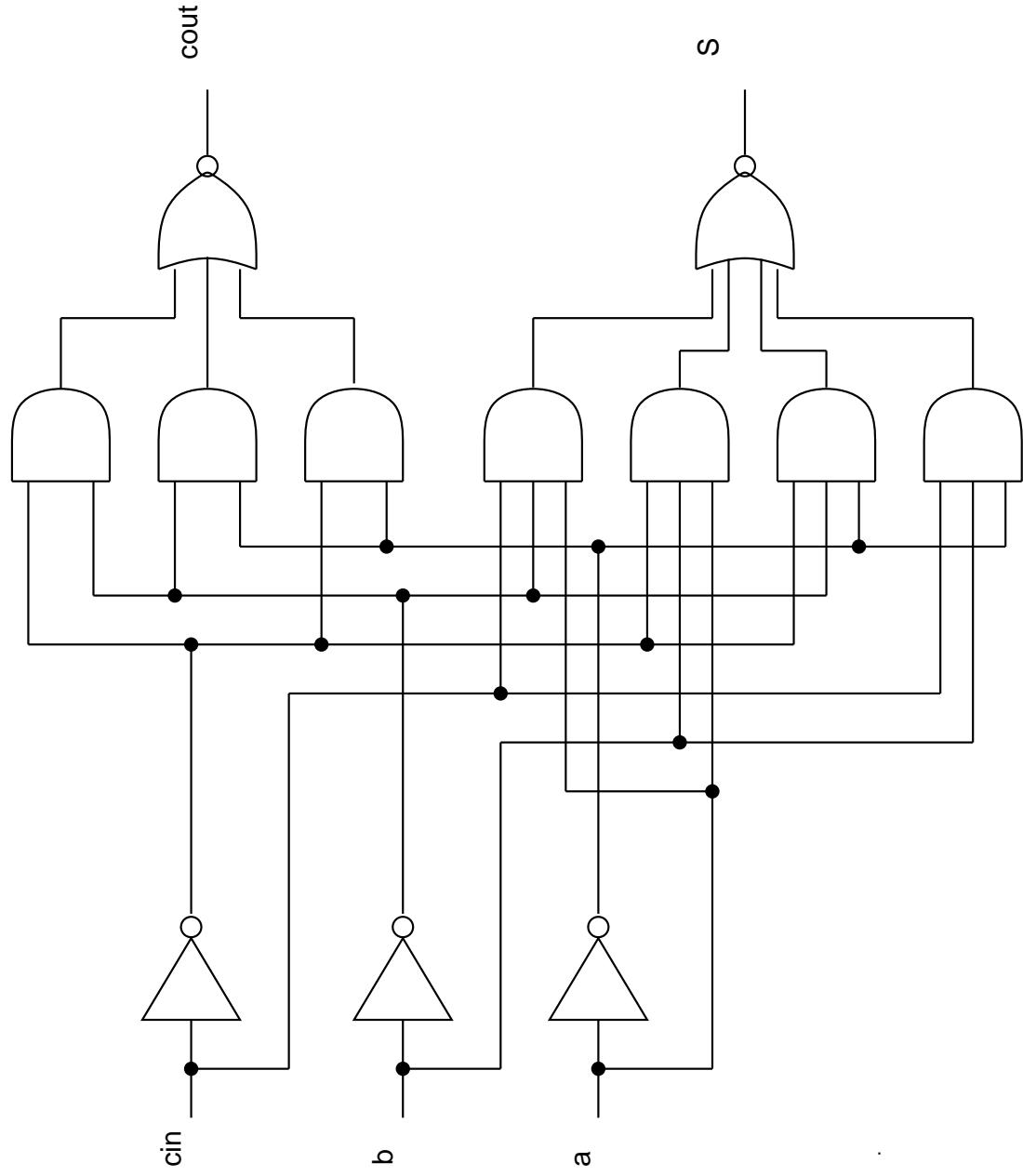
% AND3i(a,b,c) : bit =
% if a=1 AND b=1 AND c=1 then 1 else 0 endif

NOR3i(a,b,c) : bit = tf2b(NOT(b2tf(a) OR b2tf(b) OR b2tf(c)))

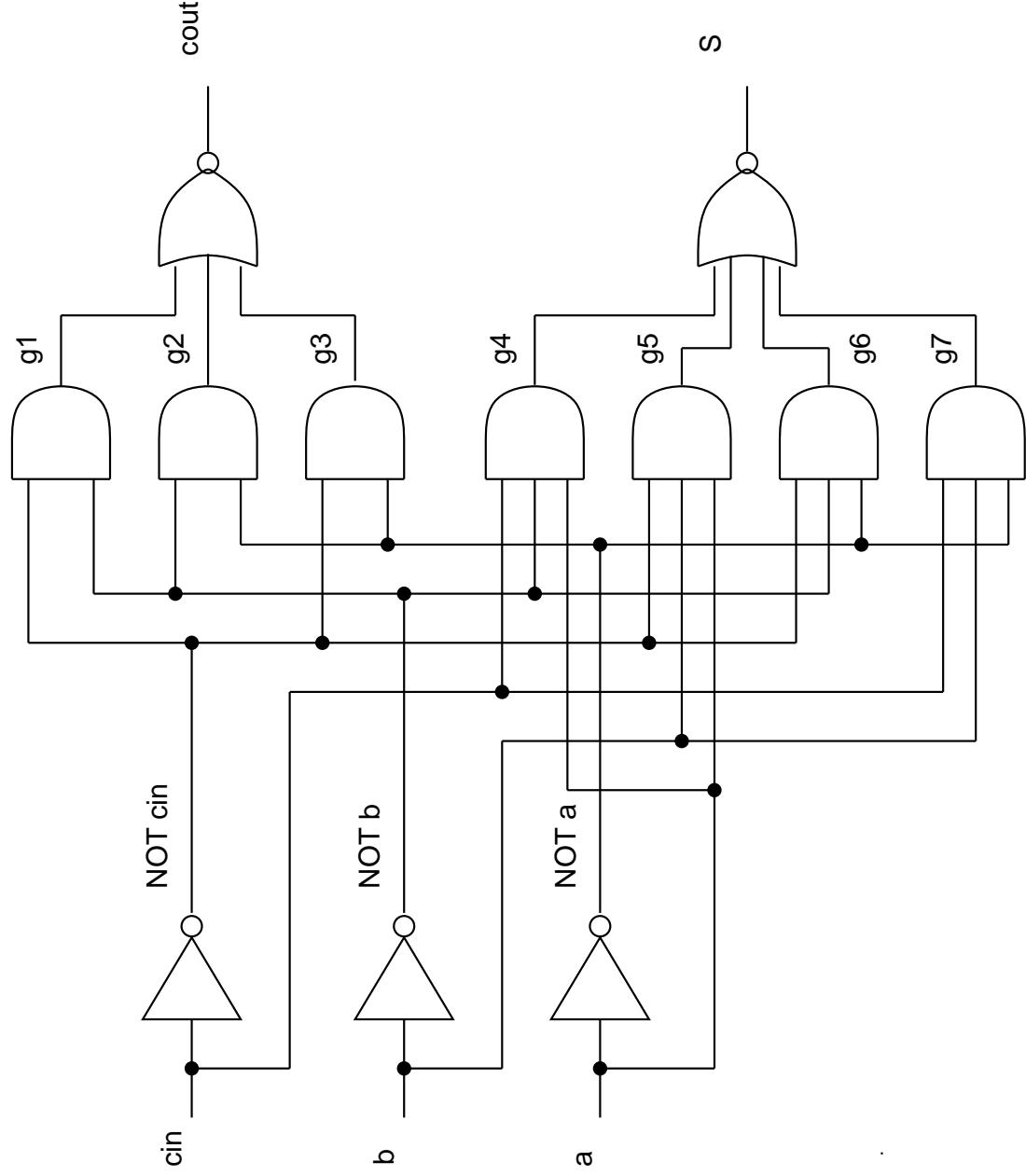
```

```
% NOR3i(a,b,c) : bit = if a=1 OR b=1 OR c=1 then 0 else 1 endif  
  
NOR4i(a,b,c,d) : bit =  
tf2b(NOT(b2tf(a) OR b2tf(b) OR b2tf(c) OR b2tf(d)))  
  
% NOR4i(a,b,c,d) : bit =  
% if a=1 OR b=1 OR c=1 OR d=1 then 0 else 1 endif
```

One Bit Full Adder



One Bit Full Adder



Gate Level Implementation

```
gate_level_impl[N:above[0]] : theory
begin
```

```
importing gates[N]
```

```
a,b,cin,adder_cin : var bit
n : var below[N]
X,Y : var bvec
```

Gate Level Implementation

One Bit Full Adder

```
gate_OBA(a,b,cin) : [bit,bit] =  
let g1 = AND2i (dNOT(cin),dNOT(b)) in  
let g2 = AND2i (dNOT(b),dNOT(a)) in  
let g3 = AND2i (dNOT(cin),dNOT(a)) in  
let g4 = AND3i (cin,dNOT(b),a) in  
let g5 = AND3i (dNOT(cin),b,a) in  
let g6 = AND3i (dNOT(cin),dNOT(b),a) in  
let g7 = AND3i (cin,b,dNOT(a)) in  
let cout = NOR3i (g1,g2,g3) in  
let S = NOR4i (g4,g5,g6,g7) in  
(S,cout)
```

Ripple Carry Adder

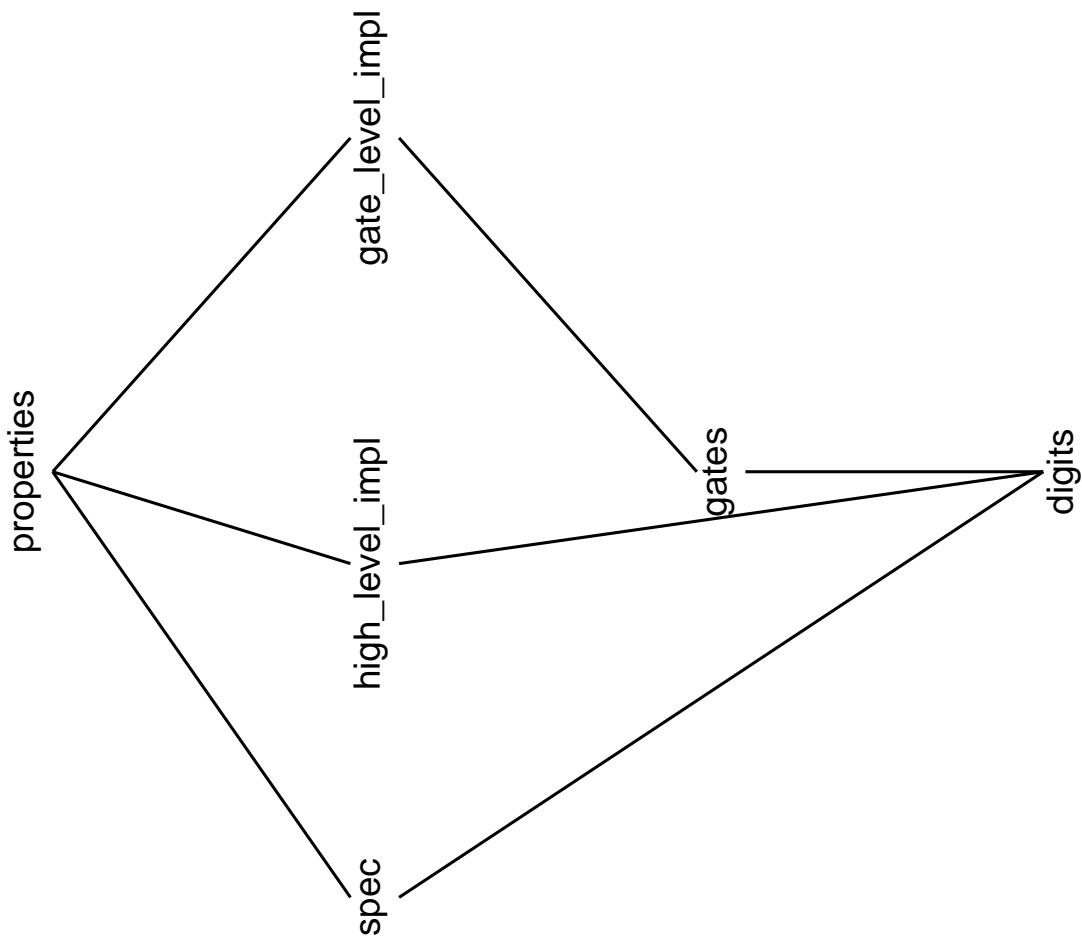
Gate Level Implementation

```
gate_adder(n,X,Y,adder_cin) : recursive [bit,bit] =  
if n=0 then gate_OBA(X(0),Y(0),adder_cin) else  
gate_OBA(X(n),Y(n),proj_2(gate_adder(n-1,X,Y,adder_cin)))  
endif  
  
measure n  
  
end gate_level_impl1
```

Properties

```
gate_OBA_correct : THEOREM  
gate_OBA(a,b,cin) = (OBA_sum(a,b,cin),OBA_cout(a,b,cin))  
  
gate_adder_coorrect : THEOREM  
gate_adder(n,X,Y,adder_cin) =  
(adder_sum(X,Y,adder_cin))(0) , n_cout(n,X,Y,adder_cin))
```

New Hierarchy



Homework

Due: 29 October 2002

- get dump file of theories and proofs.
- undump using M-x undump-pvs-files command
- prove all lemmas and theorems.
- make sure all TCCCs are proved.
- Use M-x prove-theory to prove everything and show theory status.
- dump theories and proofs onto a file using M-x dump-pvs-files.
- Mail the file to me.